# Discovery Lab 3: Deploy Advanced IOx Applications and Docker Containers

### Overview

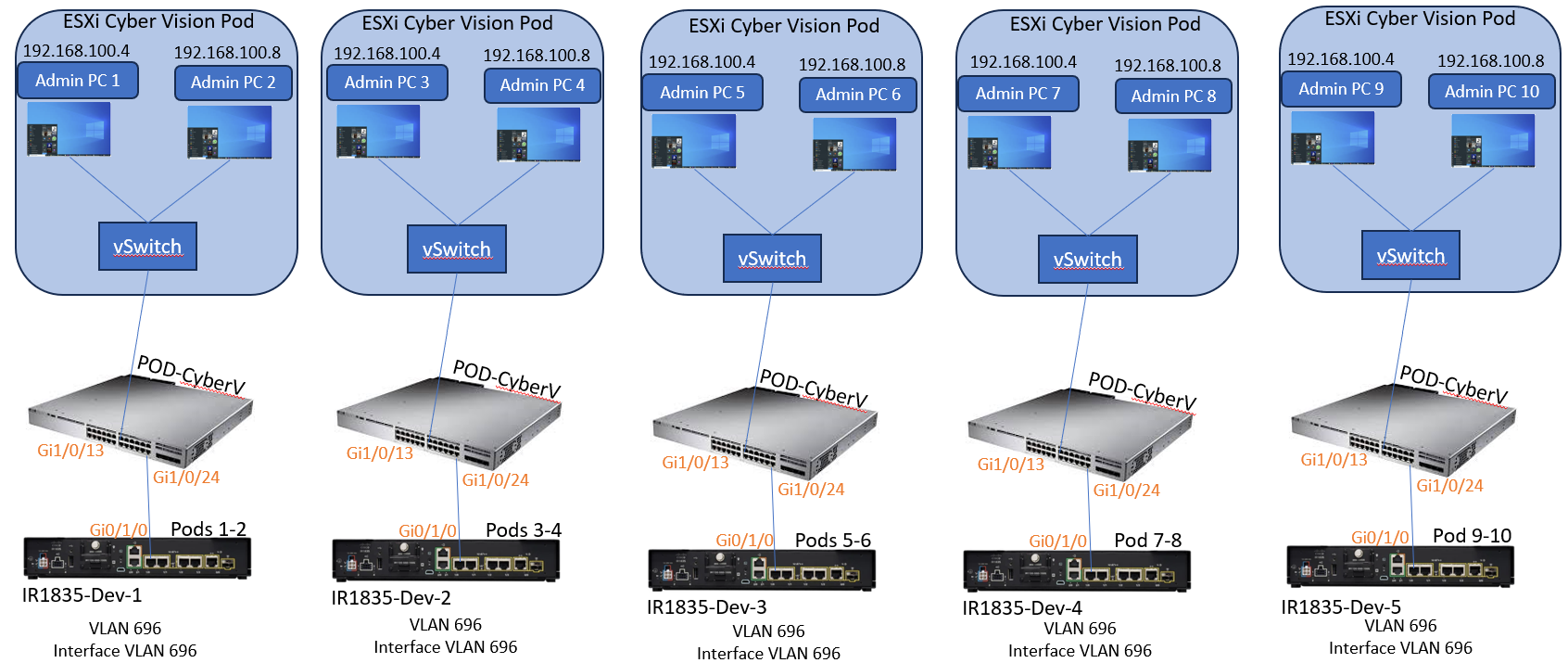
In this lab you will build and deploy custom Docker containers to be ingested by IOx in your IR1835 router. You will customize a sample web server that you can deploy as Docker container in your router. Instead of using a pre-built installation file, you will build your own docker package. For this, you need a Linux Development environment that will be provided as a hosted Linux environment in Windows (using WSL, Windows Sub-System Linux).

### Lab Topology

The following figure represents the topology used in Lab 1. 192.168.100.0/24 is a management network connecting the **Admin PCs** and the **IR1835** routers. Admin PCs are Windows 10 Virtual Machines hosted by ESXi servers. ESXi servers are connected to Cisco Catalyst 9300 Switch (identified as **POD-CyberV**). The Gi0/1/13 interface is used in that switch to connect to ESXi server. Also, in the switch the Gi1/0/24 connects to IR1835 router. VLAN 696 configured in the switch and the IR1835 router provides the IP reachability between Admin PC and IR1835 router. There are 5 independent groups of devices, each one using a separate IR1835. Every group of IR1835, switch POD-CyberV and ESXi server hosting Admin PC are in isolated management networks.

Every participant has an independent **AdminPC X** device, and 2 students share a single IR1835. Lab activities are designed so each participant will be able to do independent configurations in IR1835, without conflicts between them. Along the lab procedures, **X** represents the pod number and cab replaced by numbers 1 to 10.

In this Lab you will use a Linux Development Environment hosted by the **Admin PC X**.



## Lab Introduction

The IR1835 is a bit different in comparison with many other IOx platforms as those are mainly x86 based. The IR1835 is based on the ARM64v8 architecture so you cannot deploy containers or IOx packages built for x86 on the platform directly. This lab starts from scratch and prepares the environment for building ARM64v8-based Docker containers and explains how to build, package and deploy them on the IR1835 with the use of an x86 PC.

As an example, a very small Python script that is a simple webserver is used and a Docker container is built around to eventually package it to run on the IR1835. The only thing the webserver does is to listen on a predefined port (9000) and to return a simple web page when it receives a **GET** request. This allows you to test the capability to run your own code.

The package is built by the Docker tools, with the use of Alpine Linux. Alpine Linux is a small Linux image (around 5MB), which is often used as a base for Docker containers.

As most of the Desktop/Laptop/VMs around are all x86 based, you need to emulate the ARM64v8 architecture on the x86 based machine where the container is built. You can do this easily with the use of Quick Emulator (QEMU) user emulation. This allows execution of executables in a non-native architecture just as it would run on its native architecture.

## Task 1: Install and prepare IOx Client on the Linux Host

You need **ioxclient** in order to package the Docker container as an IOx package once it is built. The IOx client can be downloaded (with proper permissions) from Cisco Software Download web page. For your convenience, it has been downloaded and saved in the **Admin PC X**.

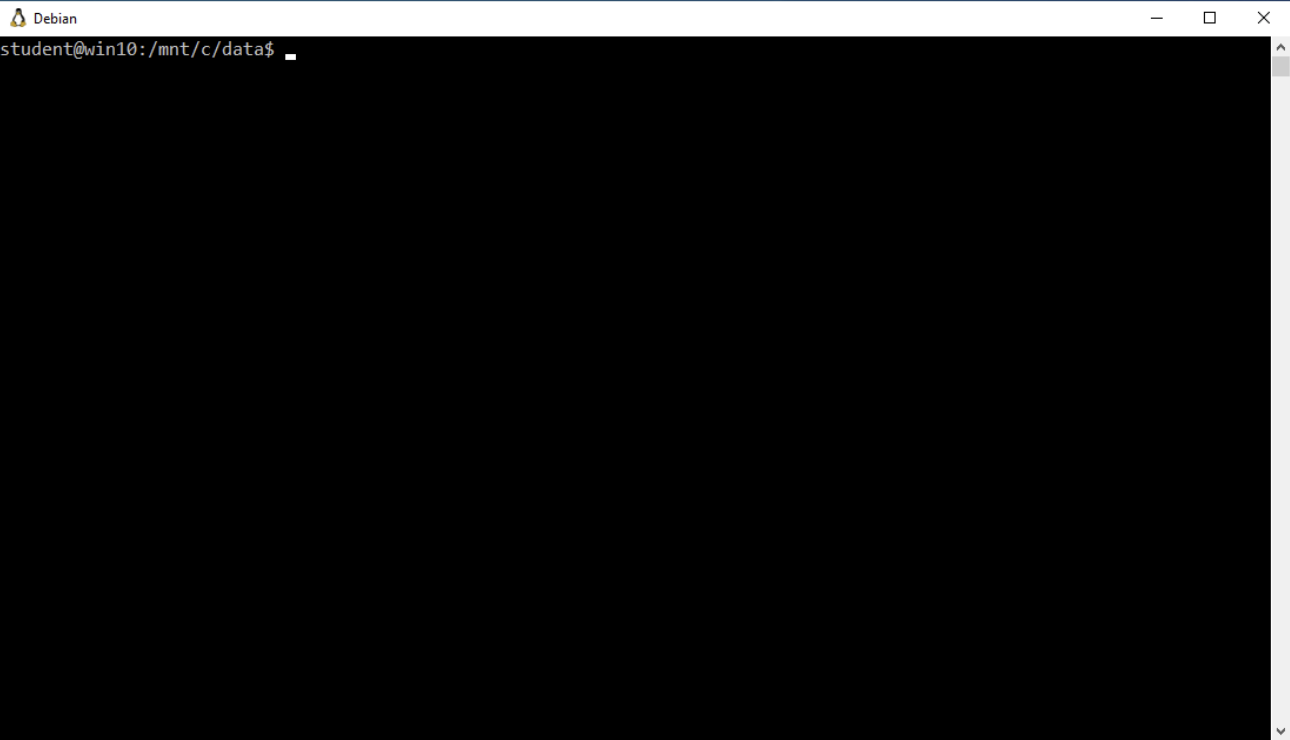
### Activity Procedure

Complete the following steps:

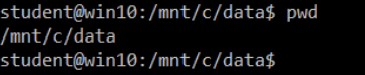
1. In the desktop of your **Admin PC X**, identify and double click the **Linux Application** (installed as part of WSL). Look for the Linux pinguin.



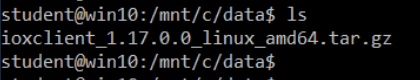
1. Linux (Debian Distribution) will start, and you will see it as a separate window.



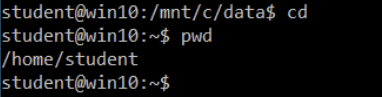
1. Check the current directory in which you are working, by typing the **pwd** linux command.



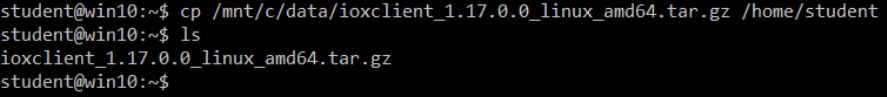
1. Use the **ls** command in linux to list the content of the current directory and confirm that the IOx client file named as **ioxclient\_1.17.0.0\_linux\_amd64.tar** is listed.



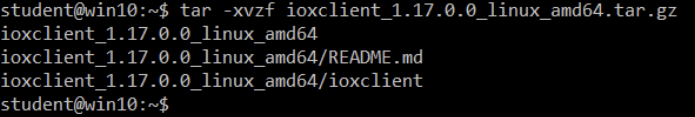
1. Change directory to work on /home/student by using the **cd** command. Then type pwd again and see that now you are in the desired folder.



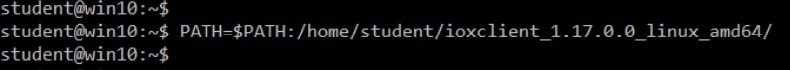
1. Copy the IOx client to this folder from its current location. Type **cp /mnt/c/data/ioxclient\_1.17.0.0\_linux\_amd64.tar.gz /home/student.** Then use again the **ls** command to confirm that the file was copied.

****

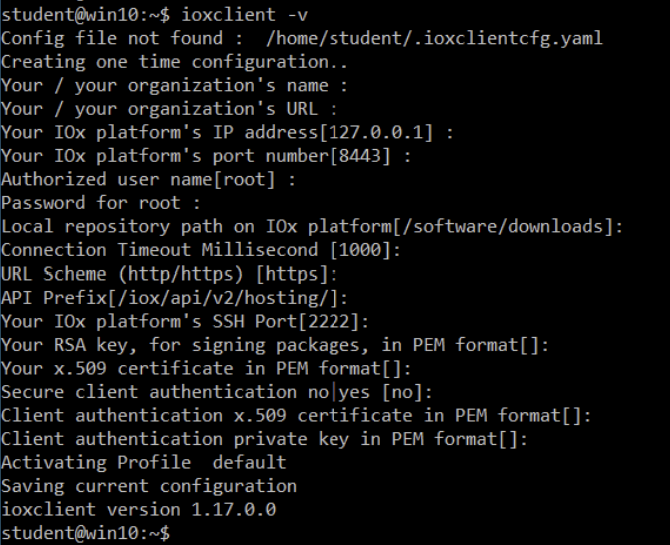
1. Extract the package by using the **tar -xvzf ioxclient\_1.17.0.0\_linux\_amd64.tar.gz** command.



1. Add the path to the **PATH** variable in order to have it available without the use of the full location. If you reboot the machine or switch users, then do not forget to repeat this step. Use the **export PATH=$PATH:/home/student/ioxclient\_1.17.0.0\_linux\_amd64/** command.



1. Launch **ioxclient** for the first time in order to create a mandatory profile. As you only use ioxclient to package the Docker container, the values can be left as default. Use the **ioxclient -v** command and press <**enter**> key. You will ne requested multiple values. Press <**enter**> key for every value requested to accept the default values.



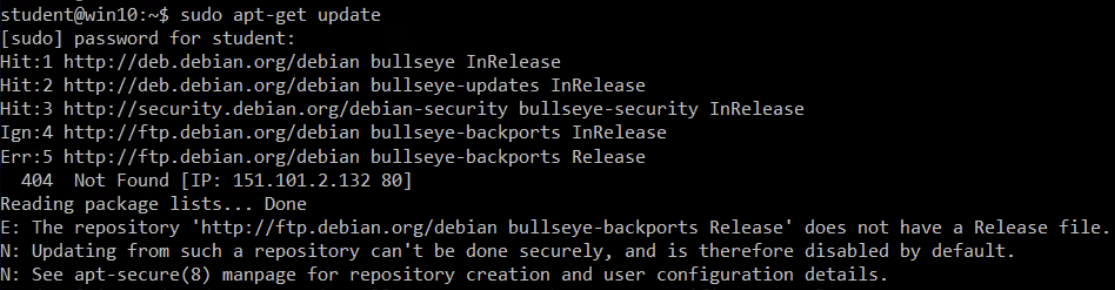
## Task 2: Install and prepare the Docker Environment on the Linux Host

### This Docker is used to build a container from the Alpine base image and to include the necessary files for the use case.

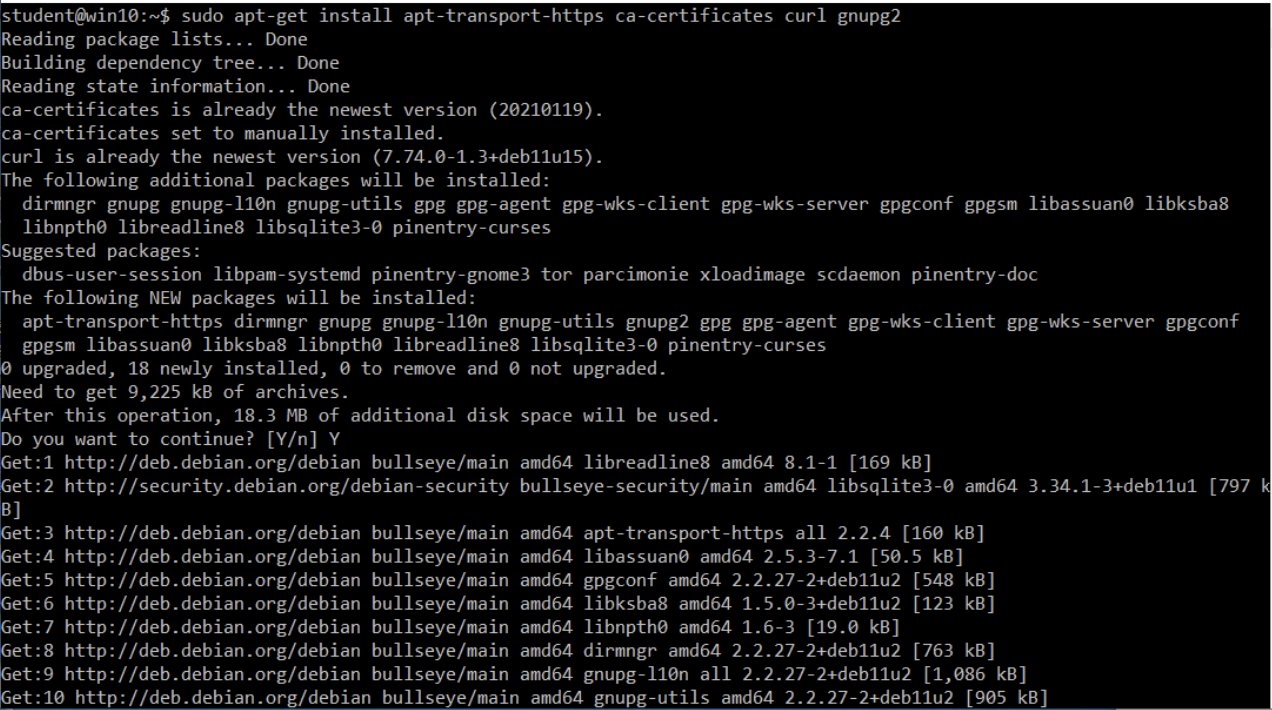
### Activity Procedure

Complete the following steps:

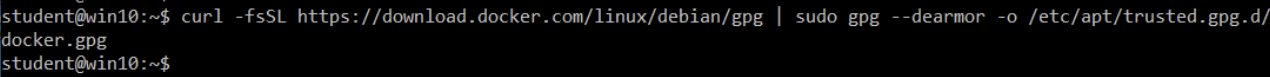
1. Update the package lists on your machine with the **sudo apt-get update** command. The sudo password is ISEisC00L.



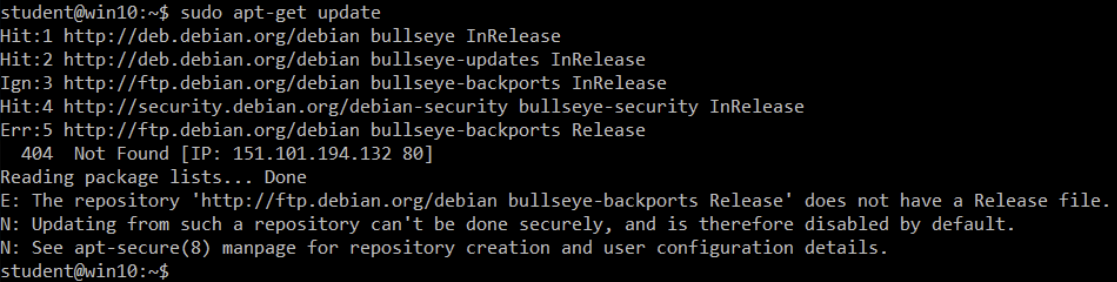
1. Install the dependencies in order to use the Docker repo with the **sudo apt-get install apt-transport-https ca-certificates curl gnupg2** command. Answer **Y** if you are asked: Do you Want to continue? And press <**enter**>



1. Add the Docker GNU Privacy Guard (GPG) key as a valid GPG key. Use the **curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /etc/apt/trusted.gpg.d/docker.gpg** command**.**



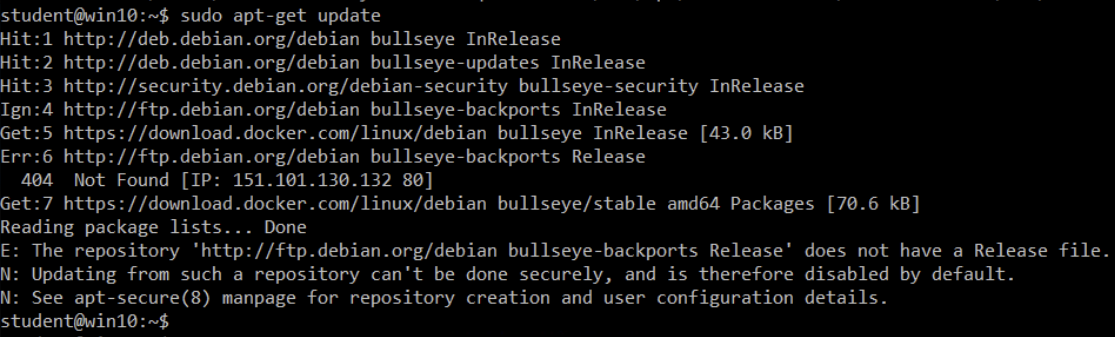
1. Update packageswith **sudo apt-get update.**

****

1. From the previous output you can see the Debian Linux Code running in this environment (**bullseye**). That name is used in the next command.
2. Add the Docker stable repo with the **echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/trusted.gpg.d/docker.gpg] https://download.docker.com/linux/debian bullseye stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null** command**.**

****

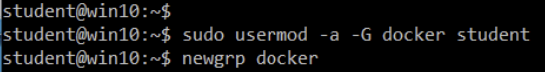
1. Update the package lists again as you added the Docker repo, using the **sudo apt-get update** command.



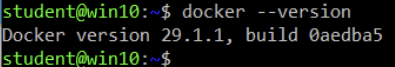
1. Install Docker by using the **sudo apt-get install docker-ce docker-ce-cli containerd.io** command. When asked, answer **Y** to continue.



1. In order to be able to access/run Docker as a regular user, add a user to the Docker group and refresh group membership. The user will be “**student**”. Use the **sudo usermod -a -G docker admin** command followed by the **newgrp docker** command.



1. Check that docker was installed by typing **docker –-version** command.



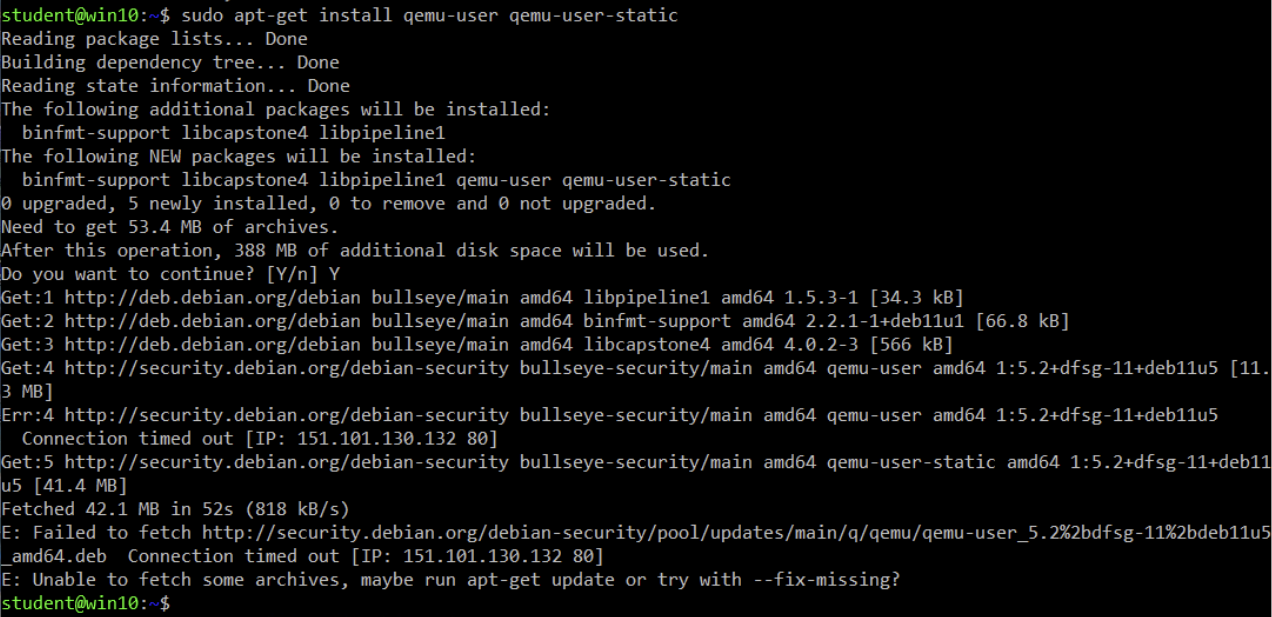
## Task 3: Install the QEMU User Emulation Packages

### After you have installed Docker, you need to install the QEMU user emulators. Use the statically linked QEMU emulator from within the Docker container so you can run the container for ARM64v8 on our x86-based Linux machine, although the target container is designed for the ARM64v8 architecture.

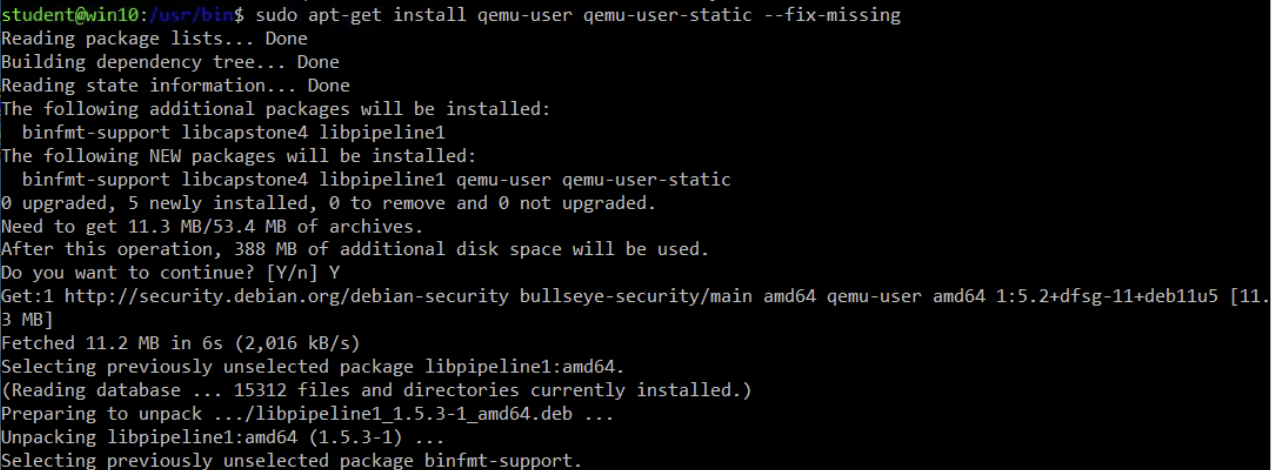
### Activity Procedure

Complete the following steps:

1. Install the packages using the **sudo apt-get install qemu-user qemu-user-static** command. Answer **Y** when you are asked to continue.



1. If you see an error message, like the one in the last line in the previous output, repeat the command, adding the **--fix missing** at the end.



1. Check the statically linked QEMU emulators available in **/usr/bin** by using the **ls -al /usr/bin/qemu-\*static** command**.** Identify those for **aarch64.** The second one in the list (**quemu-aarch64-static**), is the one you need as aarch64 is the arch-name for ARM64v8 for Linux.



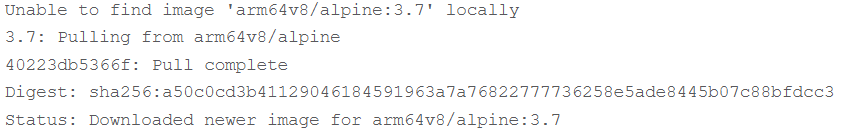
## Task 4: Test if an aarch64/ARV64v8 Container Runs on x86 Linux Machine

### Now that you have Docker and the necessary QEMU binaries installed, you can test if you are able to run a Docker container built for ARM64v8 on the x86 machine.

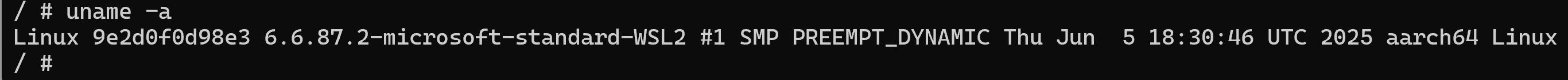
### Activity Procedure

Complete the following steps:

1. Run the Docker container built for ARM64v8 on the x86 machine by using the **docker run -v /usr/bin/qemu-aarch64-static:/usr/bin/qemu-aarch64-static --rm -ti arm64v8/alpine:3.7** command.



1. In the “**/ #**” prompt type **uname -a**.



1. In the “**/** **#**” prompt type **exit**. From the previous two outputs, you can see that arm64v8 **Alpine** container is obtained and made to run with access to the emulator. If you request the architecture of the container, you can see that the code is compiled for aarch64. Exactly as the target arch for the container must be for IR1835.

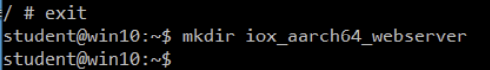
## Task 5: Prepare Files to Build the Docker Webserver Container

### Now that all preparation is done, you can go ahead and create the necessary files for the webserver container that needs to be run on IR1835. You will need to prepare two files necessary for building the container. The first file is webserver.py, the Python script which you want to run in the container. As this is just an example, obviously, you replace this with the actual code in order to run your custom IOx application. The second file that is needed is the Dockerfile. This defines how the container is built.

### Activity Procedure

Complete the following steps:

1. Create a new directory called **iox\_aarch64\_webserver**. Use the **mkdir iox\_aarch64\_webserver** command.



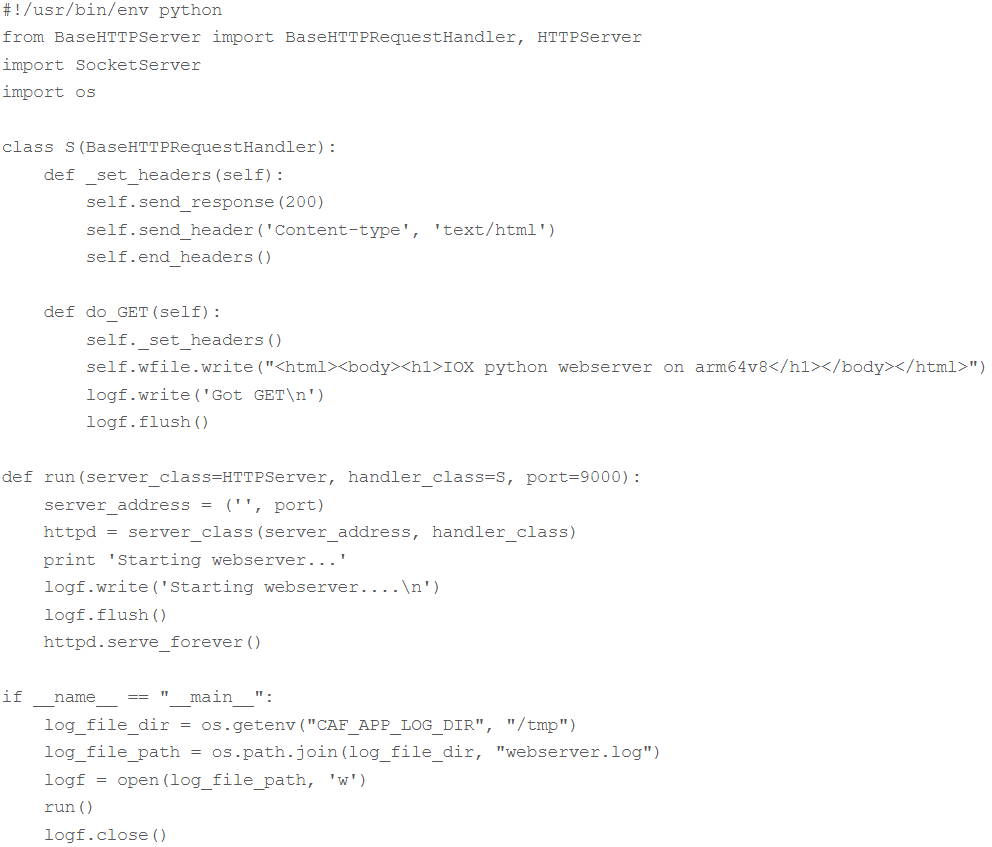
1. Change directory to work on the new created folder. Use the **cd iox\_aarch64\_webserver** command.



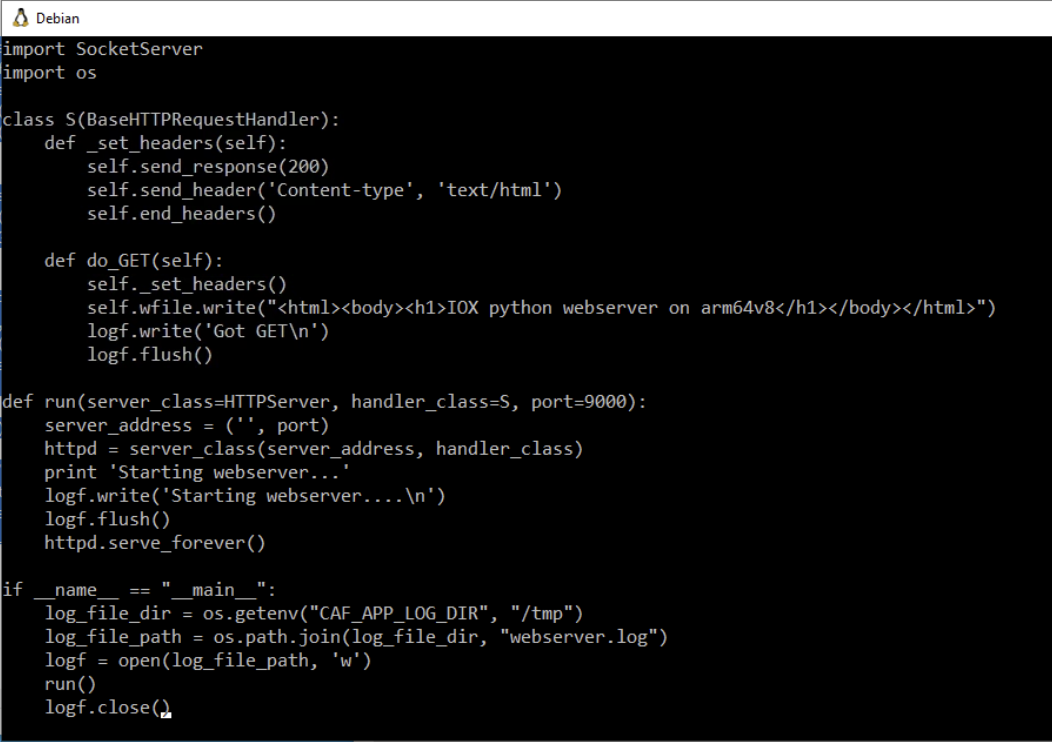
1. Use **vi** editor to create a new file with name **webserver.py**. use the **vi webserver.py** command. After pressing <**enter**> you will see an empty file.



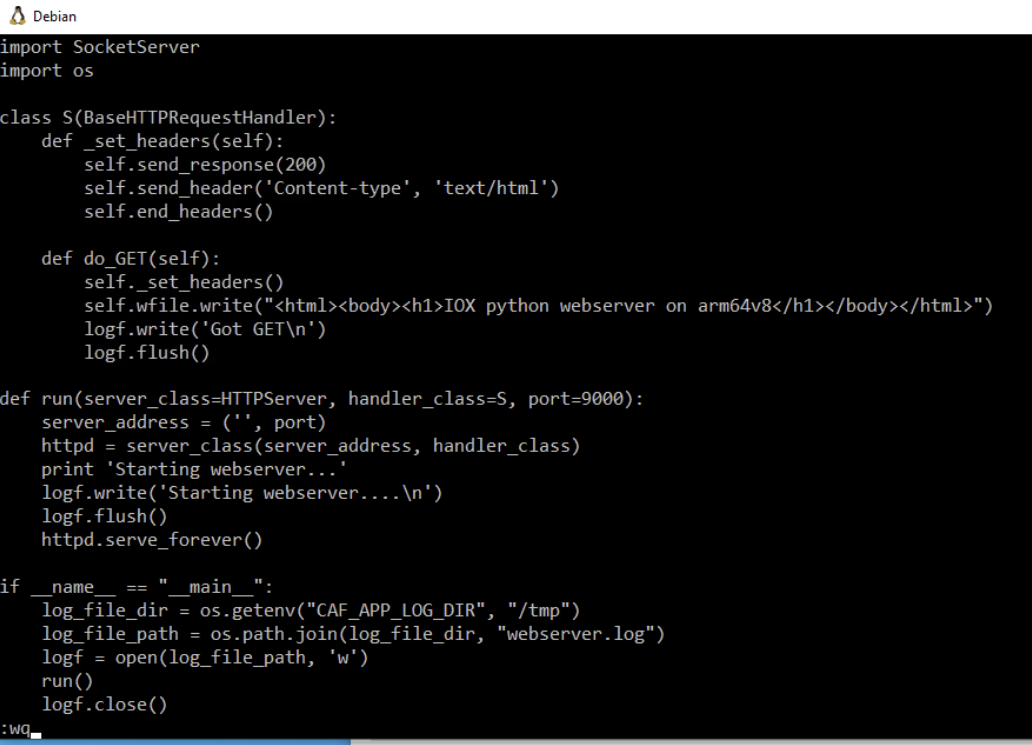
1. Yow will write (or copy from a file) the following content to the empty file. You can open the file **webserver** located in the **Admin PC X** desktop inside the **Training files/Lab 3** folder and copy/paste its content here.



1. When all the content is in the file, it will look like this:



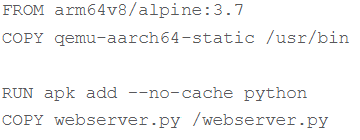
1. Press **ESC** key to exit the edit mode and enter the command mode. Then, in the lower left corner type **:wq** to save (write) and exit (quit) and press <**enter**>. You will be taken back to CLI. **HINT:** If **ESC** does not work, you can also use **CNTRL+C**.



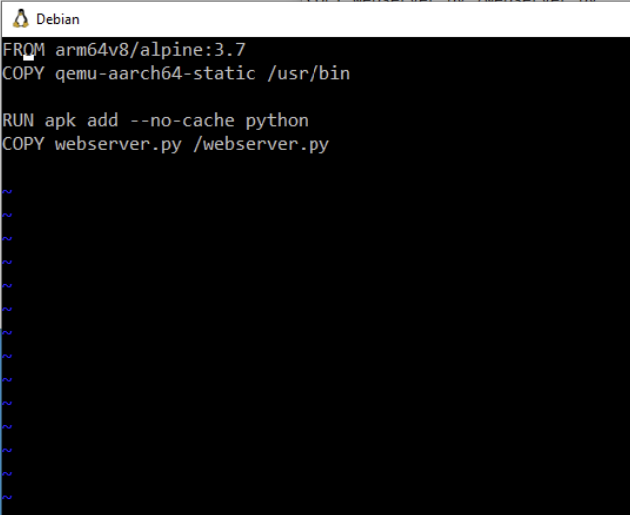
1. Use **vi** editor to create a new file with name **Dockerfile**. use the **vi Dockerfile** command. After pressing <**enter**> you will see an empty file.



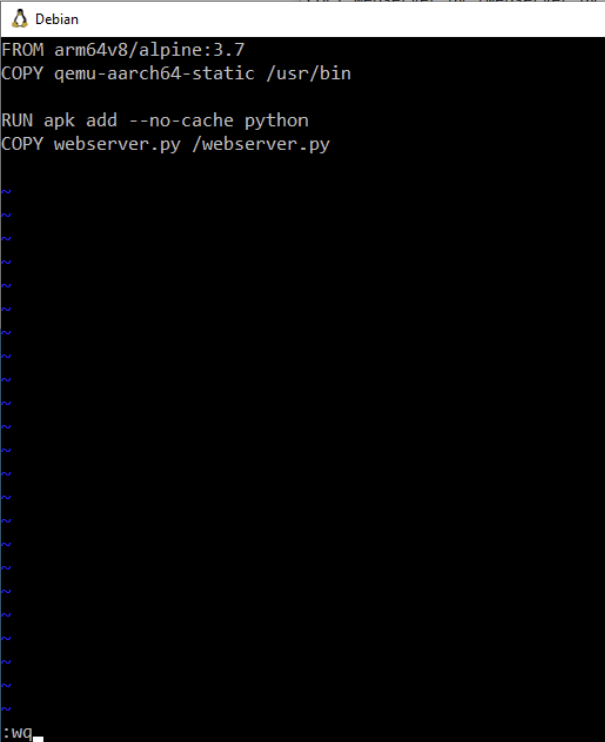
1. Yow will write (or copy from a file) the following content to the empty file. You can open the file **Dockerfile** located in the **Admin PC X** desktop inside the **Training Files/Lab 3** folder and copy/paste its content here. The Dockerfile defines how the container is built. The interpretation of the content is “Start from the **Alpine** base image for ARM64v8, copy the emulator in the container, run the apk in order to add the Python package and copy the webserver script into the container”.



1. When all the content is in the file, it will look like this:



1. Press **ESC** key to exit the edit mode and enter the command mode. Then in the lower left corner type **:wq** to save (write) and exit (quit) and press <**enter**>. You will be taken back to CLI. **HINT:** If **ESC** does not work, you can also use **CNTRL+C**.



1. Copy qemu-aarch64-static to the directory from where you build the container using the **cp /usr/bin/qemu-aarch64-static .** command (be aware that the space and dot at the end of the string are part of the command).



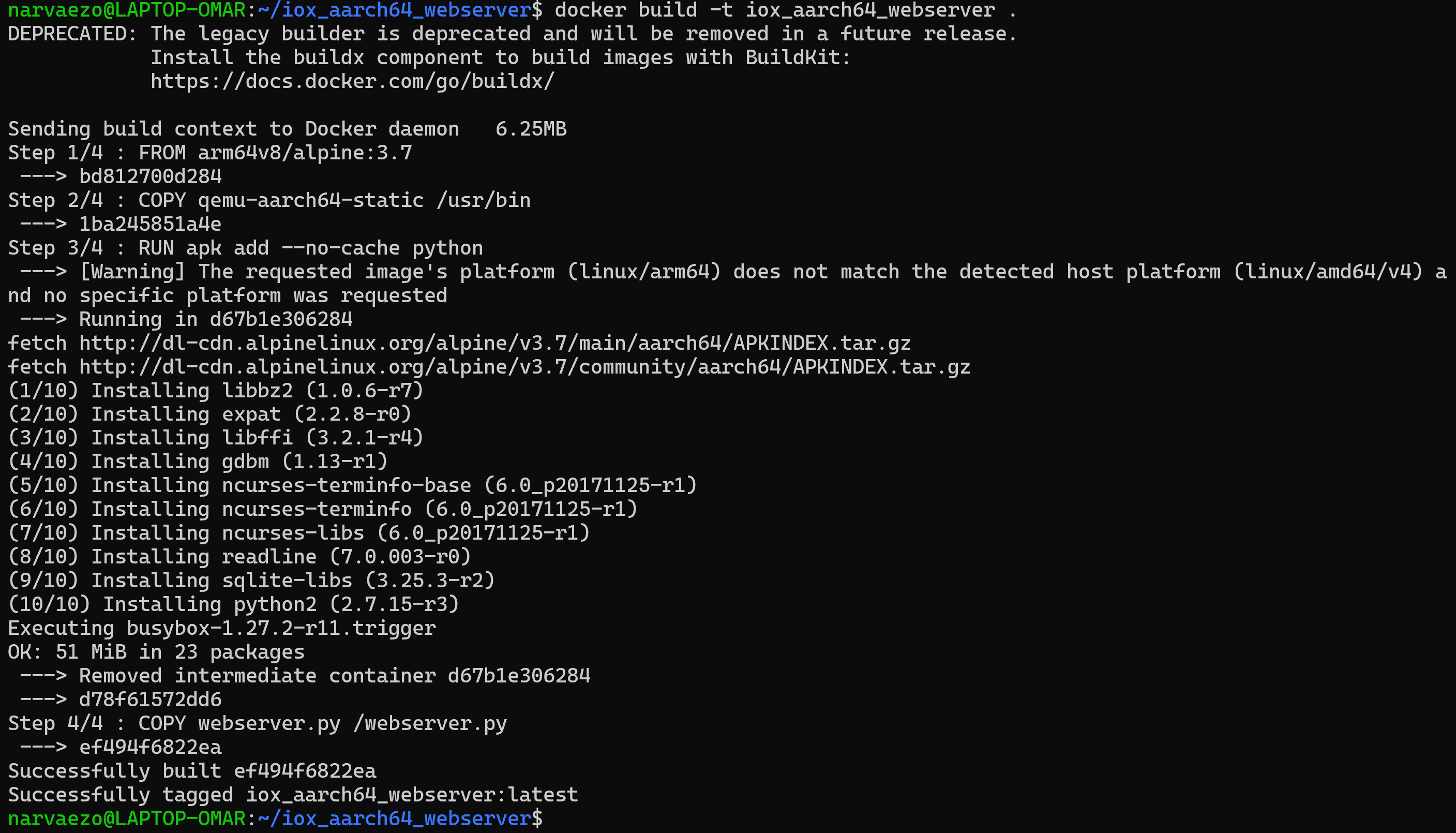
## Task 6: Build the Docker Container

### Now that all the preparation is done, you can build the container with the use of the Dockerfile.

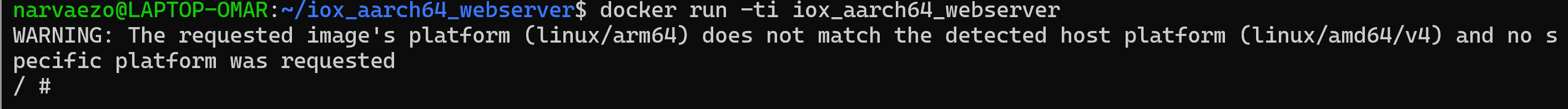
### Activity Procedure

Complete the following steps:

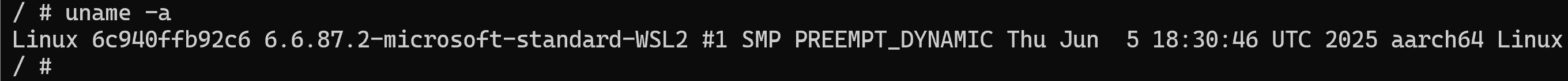
1. Build the Docker Container. Use the **docker build -t iox\_aarch64\_webserver .** command. (Be aware that the space and dot at the end of the string are part of the command).

****

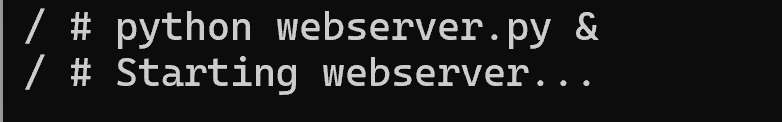
1. As a test, run the container which you just built and check if the script works. Use the **docker run -ti iox\_aarch64\_webserver** command.



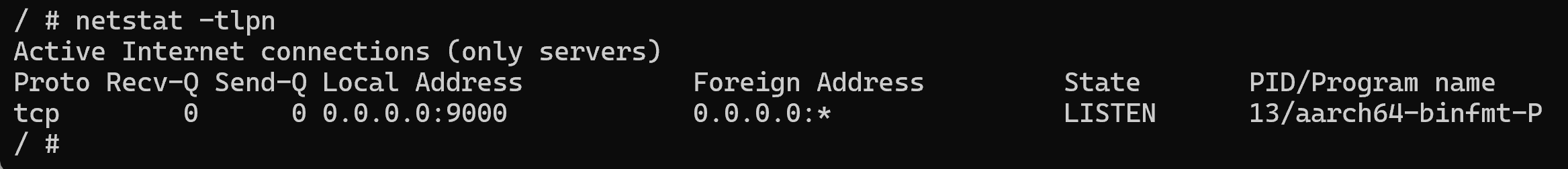
1. Type **uname -a** to continue and press <**enter**>.



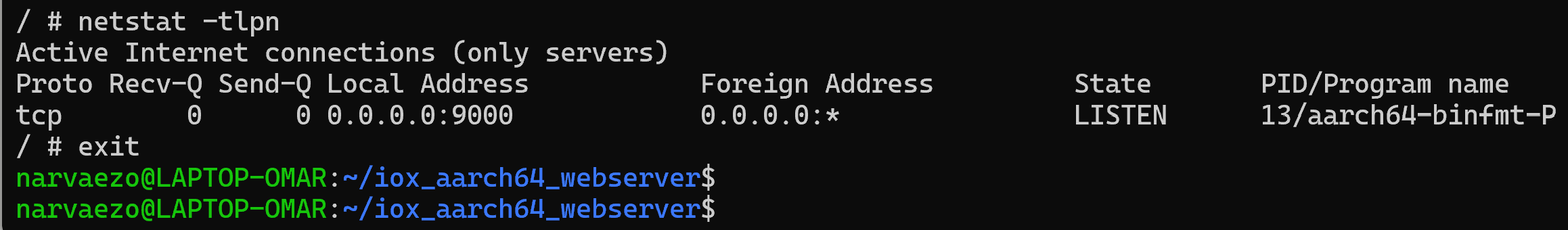
1. Type **python webserver.py &** to continue and press <**enter**>. You will see a message about the webserver starting.



1. Press <**enter**> again to display the “/#” prompt. Then, type **netstat -tlpn** command.



1. Interpret the previous script. As you can see in this output, the architecture of the container is the targeted aarch64. And after you start the script, you see it is listening for requests on TCP port 9000.
2. Type **exit**.



## Task 7: Build the IOx Package

### The container is ready to be packaged. Before you can ask ioxclient to do this, you first need to create the package descriptor: package.yaml.

### Activity Procedure

Complete the following steps:

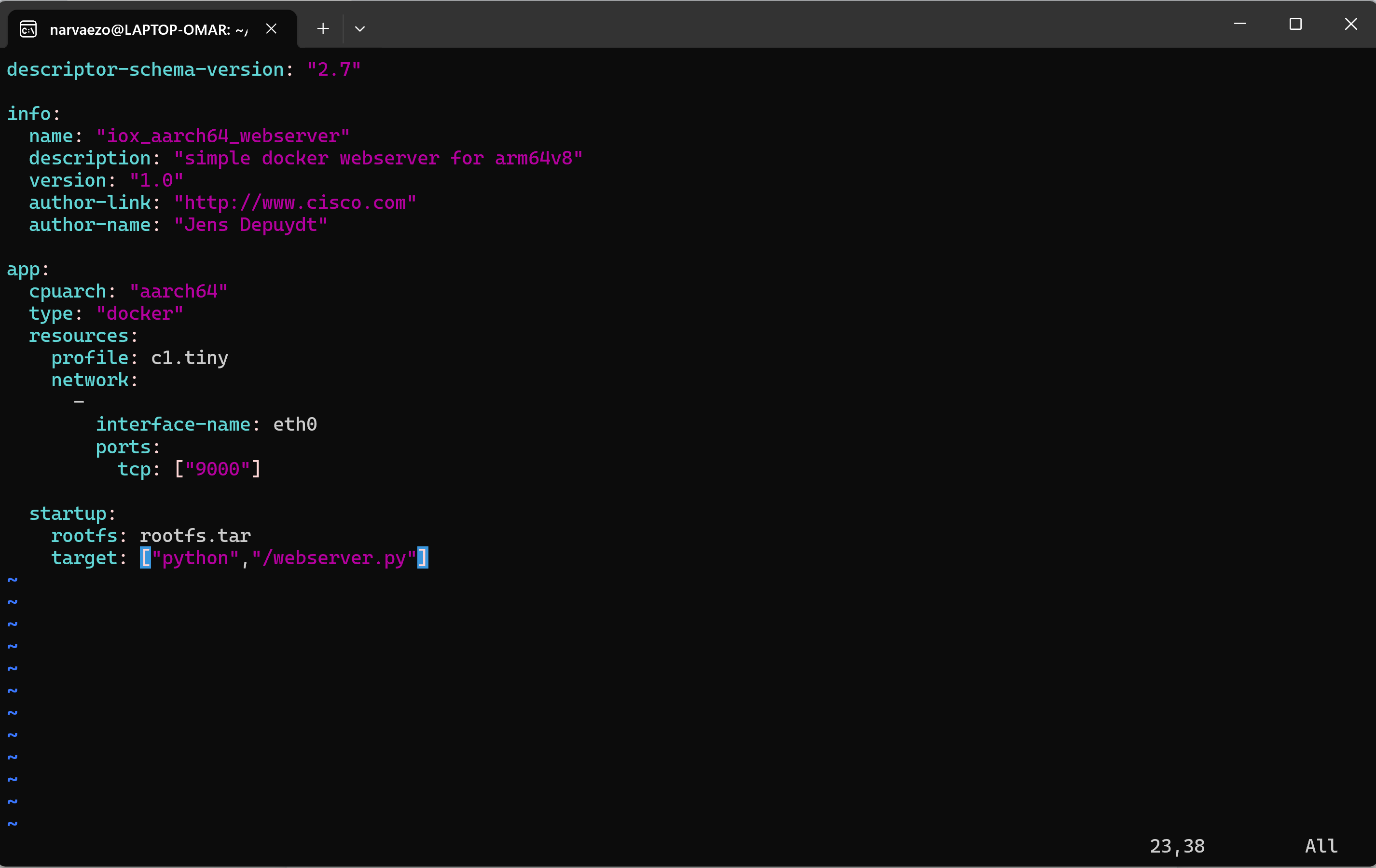
1. Build the **package.yaml** file. Use **vi** editor to create a new file with name **package.yaml**. use the **vi package.yaml** command. After pressing <**enter**> you will see an empty file



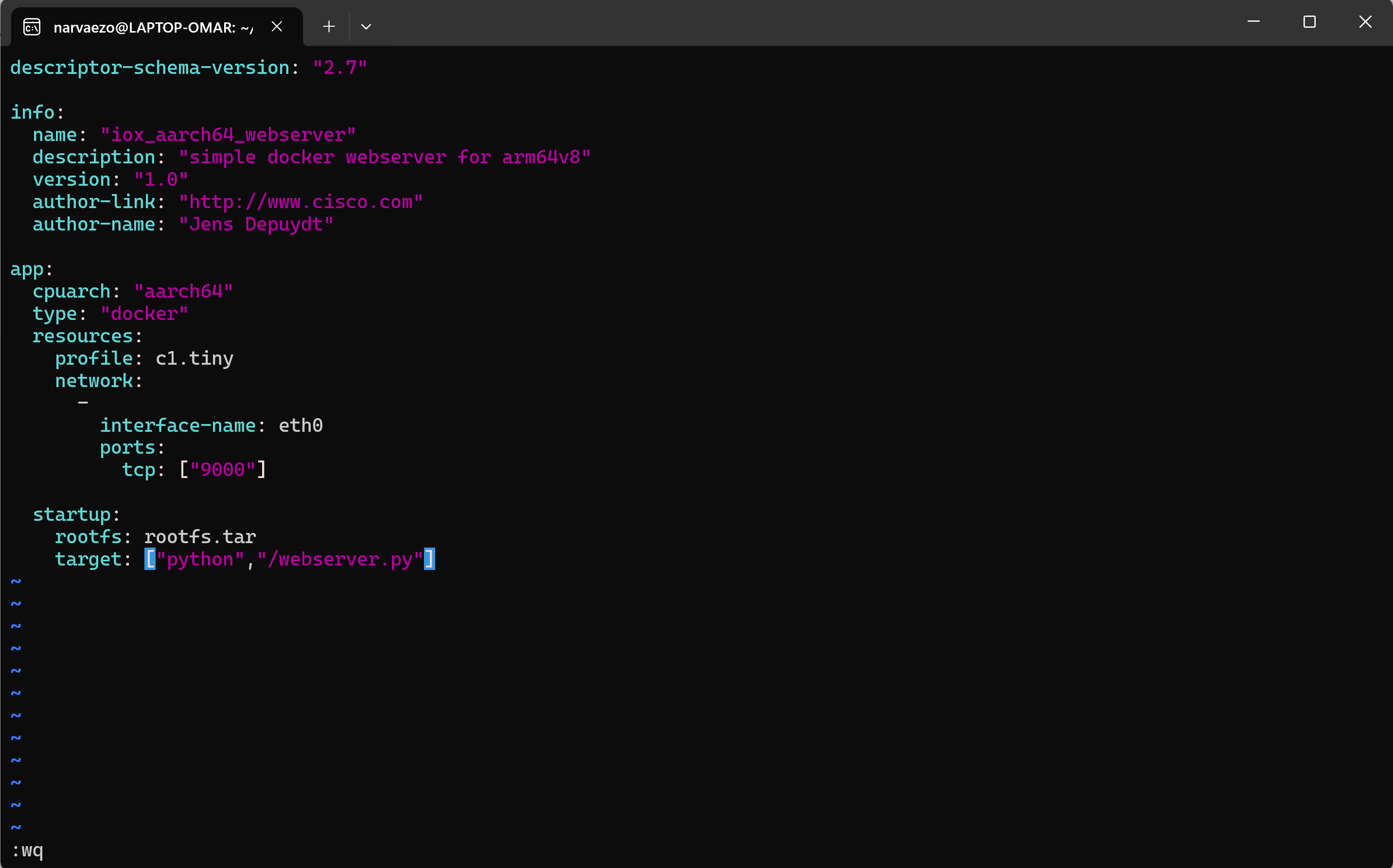
1. Yow will write (or copy from a file) the following content to the empty file. You can open the file **package** located in the **Admin PC X** desktop inside the **Training files/Lab 3** folder and copy/paste its content here. The package file describes how the package looks, how many resources it needs to run and what to start.



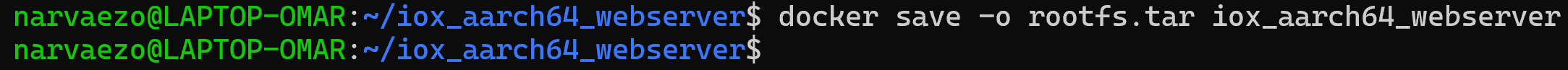
1. When all the content is in the file, it will look like the next capture. As you can see, the CPU architecture is set to aarch64. In order to gain access to TCP port 9000, use **rootfs.tar** as the rootfs and on start, you can run **python/webserver.py**



1. Press **ESC** key to exit the edit mode and enter the command mode. Then in the lower left corner type **:wq** to save (write) and exit (quit) and press <**enter**>. You will be taken back to CLI. **HINT:** If **ESC** does not work, you can also use **CNTRL+C**.



1. Extract the **rootfs.tar** from the Docker container using the **docker save -o rootfs.tar iox\_aarch64\_webserver** command.



1. Use **ioxclient** in order to build the IOx package for IR1835. For that, use the **ioxclient package .** command (be aware that the space and dot at the end of the string are part of the command). Read the last line of the output which notifies that the Docker package has been generated.



1. Using l**s** command check the list of files that have been created in the current folder, including the **package.tar**. This **package.**tar is an example of a custom docker package to deploy applications in IOx. This is the same type of file that you used in Lab 2 where the TAR files were pre-built and given to you to install the requested applications.

